# The MMSFORTH Newsletter

MILLER MICROCOMPUTER SERVICES
61 LAKE SHORE ROAD, NATICK, MASS. 01760
(617) 653-6136

January-February 1981

Vol.1 No.6




**GET YOUR FORTH NEWSLETTERS NOW!** (Editorial)

It's renewal time for the MMSFORTH Newsletter! This is Newsletter #6, the final bimonthly issue of your first $10.00 subscription. A subscription to Volume 2 - five more issues to complete 1981 - is available to our licensed users now. Send us a $10.00 check, along with your name, address, phone and MMSFORTH Serial Number. (No charge cards for just $10.00, please, but you may combine it on a larger order.) MMS aims to have information worth that amount within EACH issue. And I think we will!

The Post Office has raised some of its rates, and so has MMS. Our minimum shipping/handling charge is now $2.00, and our overseas mail rate is back up to 20%. We still refund overpayments, so add some cushion when you are not sure.

The Forth Interest Group has asked us to tell you that its newsletter has gone up in price: $10.00 for Volume 1 or Volume 2 of 4th Dimensions, $12.00 for the present year ($13.00, $13.00, & $24.00 for overseas orders). Did you listen to us and get FIG's Volume 1 when it was only $5.00?

-- A. Richard Miller, Editor 4th Class

**NEW AT MMS:**

**A BOOK FOR DESIGNING FORTH PROGRAMS:**

Are you learning Forth, but still unsure how to design your programs to take best advantage of its new capabilities? MMS is pleased to offer "Program Design and Construction", by David A. Higgins. This useful book does not discuss Forth, but does teach an easy technique for the design of structured and modular programs. We find it to be an excellent introduction to the special kind of programming best realized in the MMSFORTH environment. In lieu of a specific programming language, the book itself uses a simple and powerful program structuring notation called Warnier-Orr diagrams. A final section is written with examples in BASIC, but by then these will be easily translated into MMSFORTH by the reader. Far easier, in fact, than in the less natural BASIC!

"Program Design and Construction" is available now at MMS, for $8.95 plus our minimum $2.00 shipping/handling charge. Massachusetts customers please add the necessary 5% State tax.

**PERIPHERAL TALK**

**MMSFORTH PATCH FOR EXATRON STRINGY FLOPPY:**

On a custom basis, we've been running MMSFORTH on the tiny Exatron Stringy Floppy wafers for a year or so. This small and quick-loading peripheral is popular with computer hobbyists who are not yet ready to invest in disk drives, and it also is an excellent professional choice for certain applications. Now our Canadian MMSFORTH User Group organizer, Kalman Fejes, can provide you with his own $14.95 ESF patch (on a wafer) to merge your MMSFORTH System Cassette aboard. We haven't yet seen his routine, but we know the MMSFORTH/ESF combination can be a fascinating one. Contact Kal to order or for further information.

**SETTING FORTH** (for beginners and others)

**DATAHANDLER CUSTOM MODIFICATIONS: OVERVIEW**

THE DATAHANDLER, our database management system in MMSFORTH, is an excellent general purpose system. With a few modifications for any given task, it can become a dream come true!

MMS is very busy customizing THE DATAHANDLER for a wide variety of clients. So far we have produced dedicated versions to do specific tasks involving professional mailing labels, inventory, payroll, order entry and sales analysis, repair logging, and more. All out-perform their BASIC counterparts (often spectacularly so!) and were delivered in less time with less expense. In one of its more versatile roles, MMS has even modified THE DATAHANDLER to replace a fleet of IBM keypunch machines with one-disk Model I's for preprocessed high-volume data entry into a minicomputer system!

Other programmers as well as MMS now provide custom installation and modification of THE DATAHANDLER. Typical prices are $500 including all software and consulting time for a simple system, $1,000 for a moderately complex project. At $140, MMSFORTH and THE DATAHANDLER probably offer 90% of the final job in completed and modular form. That offers a good profit for a good Forth programmer, and could offer a full- or part-time business for you. For the client it's a good price, early delivery, and a lot of user satisfaction as well.

Because so many of you have expressed interest in this as a personal or commercial activity, MMS will explain modification procedures and key internal words for THE DATAHANDLER in this and upcoming issues.

**DATAHANDLER CUSTOM MODIFICATIONS: TECHNIQUES**

This is the article many of you have been waiting for, a first trip through the MMSFORTH source code of THE DATAHANDLER to teach it a few new tricks - and to see how MMS goes about doing such an operation.

THE PROJECT AND THE TOOLS:

We will be using MMSFORTH V1.9, its on-board full-screen editor, and THE DATAHANDLER V1.1. We will add the MERGE routine published in MMSFORTH Newsletter #1, and the MAKE-LAST routine from NL#5. In addition, we will write a new Forth routine to create custom headings on our report tables. It's a good opportunity to add and debug any MMSFORTH and DATAHANDLER fixes we've mentioned in these Newsletters, too!

This will be a complicated operation. But if you have mastered our prior beginner projects, you should be able to keep up with us.

THE GAME PLAN:

Because THE DATAHANDLER is an efficient in-memory system, it allocates to the use of its maximum-sized files just about any RAM that isn't already used by Forth and itself. This DATAHANDLER File Buffer is allocated in 1K (1,024-byte) chunks of RAM, so if we are lucky we may find additional room for our mods within the existing "last" chunk of the present program. If not, we must either trim some from the existing DATAHANDLER commands, or limit the file buffer size in order to gain RAM space for our added programming. During development, we will limit file size even further in order to run the Full-screen Editor, the .S (nondestructive stack print-out) routine, etc. After programming, testing and debugging, we will remove these extra routines and again make the file buffer size as large as possible. That's the plan, now we need only fill in the details!

DISK ORGANIZATION FOR THE DATAHANDLER:

THE DATAHANDLER source diskette we supply consists of Forth source blocks defining additional routines for various functions, then a files directory block and finally a series of blocks dedicated to the files themselves; three sample files are delivered on the diskette.

Layout of the delivered diskette is quite simple: Blocks 1 and 2 are delivered with temporary routines for adding your MMSFORTH system to this disk or a backup thereof. Your MMSFORTH system then will occupy Blocks 0-9. Blocks 10 and 11 are a disk directory like the one on your MMSFORTH system diskette, but for THE DATAHANDLER. Blocks 12 through 15 bring some necessary MMSFORTH optional routines (strings, double-precision, etc.) over for this application. THE DATAHANDLER itself begins on Block 16 and ends on Block 47. Actually, its normal mode ends on Block 45; the next two blocks define DIR-INIT, which is a special routine for creating custom DATAHANDLER Files Directory Blocks. Block 48 is a standard Files Directory Block. From there on up we have provided three sample files and then a bunch of empty, formatted blocks, on up through Block 86, the final block on a standard 35-track diskette.

(See Memory Map and Diskette Maps on Pages 2 and 3.)

Don't just take our word, see for yourself. Boot up your MMSFORTH system, then swap in your DATAHANDLER disk, enter 0 EDS and keep hitting the + key to scan through and look. While in the full-screen editor, be sure to avoid other actions which could re-write the block! Better yet, keep a write-protect tab in place during this first look.

If you don't have a merged DATAHANDLER/MMSFORTH source diskette like the one described above, merge one now according to your DATHAHANDLER instructions. Then make a BACKUP on which we will take the rest of this trip.

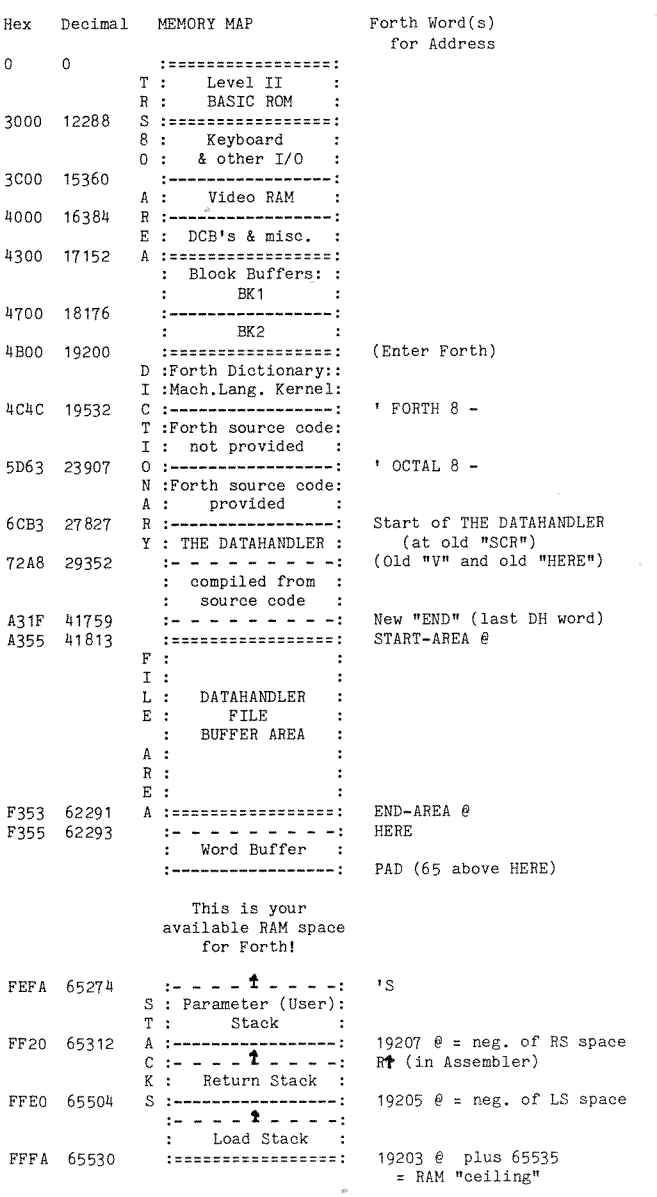
LOAN THE DATAHANDLER AN EDITOR:

It is possible to do all the editing for our new routines before loading and running THE DATAHANDLER program. But our debugging will be greatly facilitated if THE DATAHANDLER is up and

running when we will be adding new code. Actual loading of THE DATAHANDLER is directed from Block 10, and begins on Block 16. Do a 16 EDS to view this screen, and note that it first executes a FORGET SCR to remove the full-screen editor from the MMSFORTH dictionary, then loads the extra Forth definitions from Blocks 12-15, and only then proceeds to add its own new defining words. Let's undo that FORGET SCR, so we can keep the editor. While using the DATAHANDLER the editor just wastes precious RAM, but during our program development it will be a valued tool. Using the Replace submode of the Editor, put a single "(" character and a blank just ahead of the FORGET SCR on Line 0 in order to keep Forth from executing this phrase when the block later is loaded. (We could delete the offending phrase, but this works as well and leaves the code where we will see it and reactivate it when our work is completed.)

SAVE OUR SOURCE!:

Before travelling on, be sure to press S to Substitute the modified screen into the block buffer, then write your change to disk by removing the write-protect tab AND removing any software write-protection with a 0 PBLK ! (MMSFORTH uses these two separate lines of defense). Forth's virtual memory should be in control from here on in, except for the final two blocks which will require a FLUSH command. But I like to be in control here, so I "S FLUSH" each block as soon as it is ready to go to disk. If the block's code is complex, I may S FLUSH it several times to be sure I don't goof and lose the partially-written code before I'm done. The whole disk might get into trouble, too. So I remember to BACKUP the whole shooting match, at least once per session and sometimes more than that. If you ignore these safeguards once too often and pay dearly for it, you'll really appreciate just how easy and fast they would have been while you still had the chance!




**MEMORY MAP, DATAHANDLER V1.1** (with 48K RAM)

BORROW BACK SOME RAM:

Next, let's limit the maximum file size. Nothing to it! Just change the value of M#BLKS at the beginning of Block 40 Line 2. We deliver THE DATAHANDLER with this value set to 09, meaning 9K maximum file size. That's just right for a standard version running in a 32K RAM TRS-80. In a 48K RAM system, we would normally edit this value to 24; 24K maximum file size is usually plenty, and allows room for changes such as the ones we will make. But for now, let's be far more conservative and set M#BLKS to 20 for a 48K RAM system, or 5 if you will be using one with 32K RAM. We will open it again once our modifications are complete; until then, we will limit testing to the present files or relatively small new ones. Did you remember to FLUSH your changed block to disk? I'm going to trust you to do so, from here on!

WHERE TO PUT THE NEW ROUTINES?:

Where it makes the most sense, of course! For minor changes of existing routines, it's usually best to put them on the block where the routine has been. If major new sections of code are to be written, MMS likes to create a new work area starting at or about Block 80. This usually leaves adequate area for custom modification, while leaving room underneath for enough data files to test the system while it is under development. If we had to keep this area on a non-temporary basis, we would limit the files area on the disk with NEW-DIR. But it isn't necessary in this case; we will reprecompile the final system anyway, so these high-numbered source blocks will be freed for files use on our final, precompiled DATAHANDLER diskette.

First-time workers often plan to load these new blocks at the end of the standard routines. This could be a mistake, however, because THE DATAHANDLER redefines several words. One of these is END. It's the final word defined in the blocks that are loaded from source when you enter the word, DATAHANDLER. Since we may opt to use BEGIN...END constructs, it could be embarassing - or painful - to find END behaving differently at that point. Simple! BEFORE that point on Block 45, just insert a command to jump up and load the new code, then return to redefine END on Block 45. Do this by inserting one or more blank lines ahead of that END definition, and on one of them write something like:

```
80 4 LOADS  ( TO LOAD MY SPECIAL BLOCKS AT THIS POINT )
```

Assuming that you wish to create four blocks, next say:

```
EDITOR  80 CLEAR  81 CLEAR  82 CLEAR  83 CLEAR  80 EDS
```

WRITING THE NEW SOURCE CODE:

Each new block, of course, starts with a parenthesis and a 0-Line remark, which includes one's initials and the date:

```
( SPECIAL DATAHANDLER ROUTINES, PART 1 OF 4, ARM, 02/02/81 )
```

Personally, I like to begin my first new block with a set of development routines. I usually insert .S (the version which shows top of stack on right, and with KEY DROP at the end to hold until another key is pressed) and TEST. (See MMSFORTH Newsletters 1:3 and 1:4.) Other tools can be inserted here or later, according to the need.

I leave spare lines everywhere, because I needn't stint and it can make later additions easier. In Forth, good comments don't cost RAM space or execution speed either, so use them to help you compose now and to maintain later! At MMS we like to comment in lower-case for best contrast with the working code; however, we will keep all source code in caps for this article.

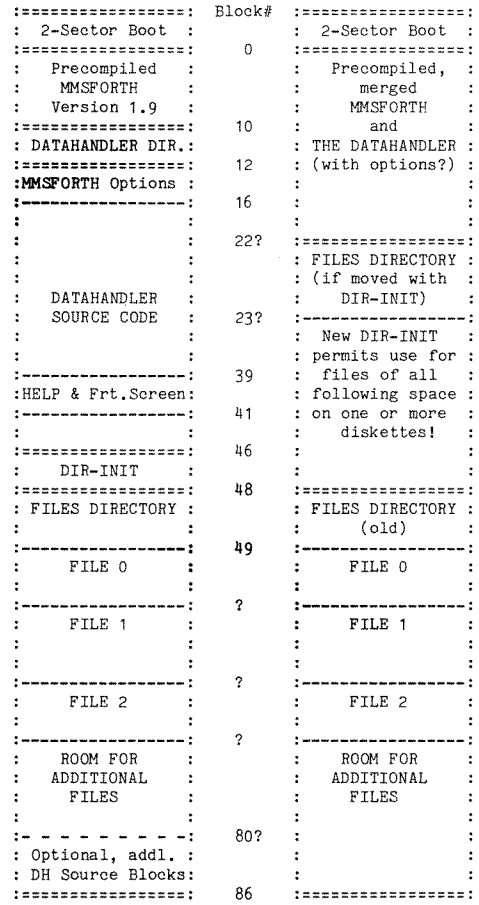
At last, we are ready to insert the new source code, starting with those development utilities, and followed by the MERGE routine. (See our comment on it elsewhere in this issue.) Key in the MAKE-LAST routine now, also. (From MMSFORTH Newsletter 1:5, but insert a "@" after RECORD-# .) By the time you've got all this plus spare lines and commenting, you will be well into a second new block. I like to bring the 0-Line along by Copying it into PAD with a shift-C, then plus-keying to the next block and replacing it into Line 0 there with a shift-R. A final R to change "Part 1" to "Part 2", and it's across without rewriting it. This is a useful and enjoyable way to gain facility with the Full-screen Editor, so try it yourself!

Since we've added all new routines for which code already exists, this seems like a good time to compile it and see how we're doing. Be sure you've written all your new blocks to disk with a FLUSH.

WHAT HATH GOT WROUGHT (OR ROTTEN)?:

Reboot your diskette, enter DIR, then enter DATAHANDLER and wait a minute to compile your new source code. When it comes up, press Break instead of 1 or 2, and then determine whether you do indeed have plenty of available RAM by entering 'S PAD - . and reading the number of bytes displayed. It should be well over 300; if not, you will have to go back to Block 40 and readjust that M#BLKS value at the beginning of Line 2, then reboot and reload. (Forth hotshots will enter FORGET BAR 40 6 LOADS instead, as we will soon see.) Now you can say DATAHANDLER again,

SOURCE DISK (w/MMSFORTH merged)  PRECOMPILED DISK (w/files)




**DISKETTE MAP, DATAHANDLER V1.1**
(Source & Precompiled, 35 tracks)

to try running with your new MERGE and MAKE-LAST routines. For now, SAVE none or only very short files, so you won't risk overwriting new file entries up into your new source code starting on Block 80. This won't be a problem on the final version.

DEBUGGING, or HAVING YOUR CAKE AND EATING IT!:

I hope they ran fine, and you are ready for the next part. But if not, debugging is easy because you now have your EDS command available while you are running THE DATAHANDLER! Just locate the offending word in its source block, edit it back into shape, and recompile ONLY the necessary blocks. If it's in one of the new blocks, catch it by entering:

FORGET CHECK  45 LOAD

See why? CHECK is the first word that was compiled in Block 45, so that strips off all higher dictionary. And 45 LOAD puts it back your new way, without wasting another full minute to recompile the lower code. Try it again, it's up and running! Pretty nice, huh?

TITLE YOUR REPORTS:

When you're through admiring your handiwork, let's get on with the last routine we wish to add, a header line for your DATAHANDLER reports. This is an original, the detailed analysis of which will await another issue (or the advanced programmer's own detective work). Here's how I chose to go about it:

```
: $VARIABLE  0 CVARIABLE H +! ;  131 $VARIABLE HEADER
: .HEAD   HEADER C@  IF HEADER $.  CR CR  2 PPOS +!  THEN ;
: SET-HEADER   CR  " OLD HEADER IS: "    HEADER $.
               CR  " NEW HEADER: "  IN$  HEADER $! ;
```

$VARIABLE was not among the string-handling functions defined in Blocks 12-15 of THE DATAHANDLER, so I have to add it here (or there). Because I want to use .HEAD both in the REPORT routine's LFS definition on Block 43 and in the REDO definition on Block 44, .HEAD must be defined earlier. But it must be defined after PPOS, which also is defined in Block 43. I could again jump from this critical point up to a higher block, but instead I chose to edit these short definitions into available space on Block 42. To do so, I found I had to shift the PPOS definition there as well, ahead of the .HEAD definition which calls it. Then we can modify the LFS definition on Block 43 Line 5, inserting .HEAD between 0 PPOS ! and L/P (moving a word or two over to the next line to make room), and we can modify the REDO definition by inserting .HEAD again, at the right end of Block 44 Line 3.

Once it's all entered and written to disk, it's again ready for recompiling. This time enter FORGET ?, 42 4 LOADS . This FORGETs the new entries into the dictionary starting at the first new definition on Block 42, then rebuilds from the newer version of Block 42 on up. Test this latest version of your masterpiece, debugging as necessary!

I used my SET-HEADER to make column headings, and found it a bit awkward because my new entries didn't start at the left margin although the finished HEADER did. So I went a step further, bringing a copy of IN$ from Block 13 to just before SET-HEADER. By renaming it to SPEC-IN$ and fiddled slightly with it and SET-HEADER, I got SET-HEADER to begin both versions of HEADER at the CRT's left margin. If you are feeling up to this, try it yourself as an advanced exercise until our next issue.

WRAP IT TO TAKE OUT:

Have you thoroughly adjusted and debugged your routines? (Has ANYONE, EVER?) When you're ready to stop modifying your code, put on the finishing touches. On the bottom of Block 39, remove the semi-colon at the end of the present MENU definition and add another line, perhaps like this:

```
CR CR " SPECIAL:  MAKE-LAST      MERGE      SET-HEADER" ;
```

On Line 2 of Block 40, reset the higher value for M#BLKS, the maximum number of blocks of file that can be buffered. And remove the paren from ahead of FORGET SCR , on Block 16 Line 0. Up on Block 80, insert an open parenthesis at the beginning of each line of the temporary development utility words: SO and .S , etc. (This way, they won't be compiled on the next pass, but are sitting where you'll want them sometime in the future.)

MEASURE IT FOR SIZE:

Did we miss anything? You are about to remove that Full-screen Editor from THE DATAHANDLER again, so think now! If you're done, FLUSH to be sure all this good stuff is on the diskette. Then reboot, DIR, and DATAHANDLER for another one-minute load operation of the whole shooting match. When it's loaded, press Break again, enter 'S PAD - . to read the bytes of available RAM, and ascertain that it is more than 300 and not more than about 1500. If the figure is too small, reboot, 40 EDS, and reduce M#BLKS by 1 for each 1024 bytes it was under. Or if it is too large, reboot, 40 EDS, and increase M#BLKS similarly. Then you will have to 0 PBLK ! FLUSH, DIR and DATAHANDLER as before to get back to this point.

We're not done measuring yet. Once the proper maximum filesize (M#BLKS) is determined, we also have to decide how high the compiled code will stretch on the disk. Not up to Block 47 like the source code, certainly! To find out, follow the precompiling steps in your DATAHANDLER instructions. Note that in THE DATAHANDLER, the number of bytes to be reprecompiled from RAM is START-AREA @ 19200 - instead of the larger HERE 19200 - , since there is no need to save the DATAHANDLER File Buffer Area's information here. (Each appropriate file's blocks will be brought in when needed.) If our compiled code fills 84 sectors plus just a little of the next, we need 85 sectors and that means we need 22 complete blocks (which is actually 88 sectors) to do the job. Since they will start with Block 0, they will stretch through Block 21 and Block 22 will become the lowest block available for files; i.e., the files directory block. Remember those numbers: 88 sectors, and Block 22.

Reboot once again, and replace the 48 on Block 40 Line 15 with the 22 we just calculated. This will become the new files directory block, and also the new location for PBLK in the new DATAHANDLER system; that is to say, the lowest block on which we will permit disk rewrite during use.

REPRECOMPILE:

BACKUP this final source disk onto another, then reload THE DATAHANDLER from source one last time. Again press Break, and then follow DATAHANDLER instructions to reprecompile this RAM code to disk. Also reset the number of sectors to be booted, as explained there.

We now have a diskette which will QUICKLY boot up our customized DATAHANDLER, but it is trained to expect a Files Directory Block on Block 22, and we still have only the original one up on Block 48. This is where DIR-INIT comes in. Follow its directions in your DATAHANDLER instructions to create a new Files Directory Block on Block 22 which will span the rest of your Drive 0. Then use NEW-DIR to move the files across, one at a time, from the now temporary Block 48 directory to the new one on 22. At last, you are done! Now you know some of what we get paid for, here at MMS. With experience, it can become easy for you, too.

Have fun with this custom programming project. Don't forget to renew your subscription to this Newsletter, because the next issue will feature our expose of key DATAHANDLER internal words!

## MMSFORTH MODIFICATIONS

**MAKE-LAST FIX:**

Our thanks to Nick Reinhardt of Lexington, Mass. for reporting an error-producing typo in our last issue. In the first line of the MAKE-LAST definition, a "@" character was accidentally deleted. Fetch it back, so the line reads:
: MAKE-LAST ONE IF IRECORD DUP RECORD-# @ 1- IRECORD DUP @

**MERGE MYSTERY:**

Our MERGE command for THE DATAHANDLER (Vol.1:2 of this Newsletter) does fine in actual situations, but we have detected a strange effect when demonstrating it by "doubling" an existing file; i.e., MERGEing it with itself. If the original file was small enough the doubled file has twice its number of records, as expected. However, SAVEing and then MERGEing the file a second time only produces three times the original number of records, instead of the expected four! The apparent problem only occurs when MERGEing the same data twice from fewer than three file blocks, so that the newly-saved file doesn't get re-read into the two block buffers from diskette (because THE DATAHANDLER thinks it's already in the block buffers).

If this paradox disturbs you, just modify the MERGE routine by inserting the words FLUSH ERASE-CORE on Line 10 between MERGE and CLS.

**NEW JKL BLOCK:**

The TRS-80 Model III is incompatible with the way the present MMSFORTH JKL routine, on Block 34, uses PAD. Also, MMS has been exercising the many options of the Epson MX-80, an excellent and inexpensive new dot-matrix printer which we are using and selling. It has a mode in which it accepts TRS-80 graphics characters, as does the Okidata Microline-80. We prefer its normal mode which has many other options, but in this mode each TRS-80 graphics code must be increased by 32 decimal in order to print the proper character.

Our replacement Block 34 solves both problems, elegantly!:

```
BLOCK : 34

 0 ( JKL-TO-PRINTER ROUTINE )     DECIMAL
 1
 2 : ALFA   OVER +  SWAP     ( STARTING ADDRESS   # OF BYTES   ALFA
 3   DO  I C@  DUP 127 >  OVER 192 <  AND   ( TRS-80 GRAPHICS CODE?
 4      IF  32 +       ( <-- CHANGE THIS LINE:  "IF  32 +" FOR EPSON
 5                ( PRINTER, "IF" FOR OKIDATA, ELSE "IF  DROP  46".
 6      ELSE  DUP 32 <  OVER 191 >  OR
 7         IF  DROP 46  THEN ( IF UNPRINTABLE, SUBSTITUTE A PERIOD.
 8      THEN  ECHO
 9   LOOP ;
10
11 : JKL   PRINT  15360  16 0
12   DO  DUP  64 -TRAILING  ALFA  CR  64 +
13   LOOP  DROP  CR CR CR CR CR CR  CRT   %CONT ;
14
15 ' JKL INTRP !
```

**STRETCH YOUR STRINGS WITH IN$ AND $COMPARE:**

Earlier versions of IN$ may be identified as a single line of Forth source code which appears on Block 30 Line 1 of the MMSFORTH System, and on Block 13 Line 4 of THE DATAHANDLER. These versions are not suitable for strings longer than 64 bytes. Substitute this two-line version of IN$ to handle up to 255 bytes at a bite, instead:

```
: IN$  "  ? " PAD 255 EXPECT 1 PAD TOKEN DROP HERE C@ HERE +
  PAD HERE C@ + HERE C@ 1+ -MOVE PAD ;
```

To find room in Block 30, you can Delete Block 29 Line 1 and then move $XCHG to Block 29 Line 15. (To be neat, relabel the 0-lines accordingly.) We made room in Block 13 by moving the $.R definition up alongside $.L - dirty, perhaps, but effective.

$COMPARE, defined on the same blocks as IN$ in both MMSFORTH and THE DATAHANDLER, also will need modification to compare strings longer than 128 bytes. Find the five-line definition beginning CODE $COMPARE, and change its second and third lines to read as follows:

```
 C INR B INR  BEGIN HL INX DE INX C DCR =0 IF B DCR 0 HL LXI
 BC POP =0 IF PSH THEN HL DCX PSH THEN B DCR =0 IF 1 HL LXI
```

You're done, but in 16K RAM this slightly larger code will overcrowd the SORT demo program on earlier copies of V1.9. Avoid the crunch by having SORT temporarily FORGET SCR, just ahead of
: TASK ; on Block 45 Line 0. And put back the editor later, by ending Block 44 Line 15 with:
FORGET TASK 22 3 HEX LOADS DIR .

**IF YOU ARE NOT UP TO LOWER CASE:**

Well, not everyone has it yet! MMS recommends that you upgrade your TRS-80 Model I to support lower-case characters, preferably via Radio Shack's own upper-lower case hardware modification to the keyboard unit. Commencing with our Version 1.9 this mod is supported with a upper-lower case driver routine, written in Forth and incorporated into the precompiled MMSFORTH that comes up upon booting the system.

But if for reasons of your own your hardware doesn't support our lower-case driver, you have several options. You can keep it aboard and attempt to ignore the occasional "garbage" characters which show up instead of lower-case text, or you can use the ALL-CAPS routine included on the UTILITIES DISKETTE to rework the text on the diskette before attempting to work with it in RAM.

Perhaps the simplest and most overlooked approach to this problem is to reprecompile the Forth system, after removing the command which loads its lower-case driver block. This is Block 66, and it is invoked from Block 18 Line 14. (Block 67's full-ASCII keyboard driver is invoked simultaneously, and can be kept or removed at this time.) Change the 66 2 LOADS to 67 LOAD, or delete it completely to make both routines off limits. Also be sure to add the following new definition of CRT, ahead of PRINT. (The old one was on Block 66.) On the same Block 18 is a good place to do it:

```
4BD0 CONSTANT CCRT    : CRT  CCRT OUT-UNIT ! ;
```

Then reprecompile with:
HEX FORGET OCTAL 0C 0D LOADS DIR CUSTOMIZE
to create a modified MMSFORTH System which sees all lower-case letters as their upper-case counterparts.

## FUN & GAMES

**LIFE PATTERN:**

Try this square and unusually long-lived LIFE pattern. Begin by Moving down to X=24,Y=30. Then Draw 6 to the right, 12 up, 12 to the left, 12 down, and 6 to the right. Press G to generate the LIFE pattern!

**THE LAST WORD:** "Your FORTH language is Maravilhosa!"
- Edgar Pullen, a user in Brazil